A Reed-Solomon Decoding Program for Correcting Both Errors and Erasures

R. L. Miller, T. K. Truong, and B. Benjauthrit
Communication Systems Research Section

I. S. Reed
Department of Electrical Engineering
University of Southern California

This article discusses the software implementation of a simplified algorithm for decoding errors and erasures of Reed-Solomon (R-S) code words using the techniques of finite field transforms and continued fractions. In particular, random code words from the (255, 223, 33) R-S code over $GF(2^8)$ are corrupted by random error and erasure patterns, and decoded whenever theoretically possible. A matrix of execution times of this decoder under varying sets of errors and erasure patterns is also included. This matrix demonstrates the relative amounts of time required for decoding different error and erasure patterns, as well as the correctness of the algorithms and the software implementation.

I. Introduction

Recently the authors in Ref. 1 developed a simplified algorithm for correcting erasures and errors of Reed-Solomon (R-S) codes over the finite field $GF(p^m)$, where p is a prime and m is an integer. For a space communication link, it was shown (Ref. 2) that a 16-error-correcting R-S code of 255 8-bit symbols, concatenated with a $k = 7$, rate = 1/2 or 1/3, Viterbi decoded convolutional code, can be used to reduce the signal-to-noise ratio required to meet a specified bit-error rate. Such a concatenated code is being considered for the Galileo project and the NASA End-to-End Data System.

In such a concatenated code, the inner convolutional decoder is sometimes able to find only two or more equally probable symbols. Then the best policy is to declare an erasure of the symbol. If the outer R-S code is able to use the additional information that erasures have occurred, then it is reasonable to assume that the system performance will be enhanced.

In this article, an algorithm based on the ideas in Ref. 1 is used to correct erasures and errors of R-S code words using a finite field transform over $GF(2^8)$. This algorithm is written in FORTRAN V and is implemented on the UNIVAC 1108 computer. A matrix of decoding times for correcting errors and erasures of the code words is given at the end of this paper.

II. The R-S Decoding Algorithm

The algorithm given in (Ref. 1) was used to correct patterns of t errors and s erasures of the words of the (255, 223, 33) R-S code, where $2t + s < 33$ and the symbols belong to the finite field $GF(2^8)$. Define the following five vectors:

$$(c_0, c_1, \cdots, c_{254}) = \mathbf{c}, \text{ code vector}$$

$$(r_0, r_1, \cdots, r_{254}) = \mathbf{r}, \text{ received vector}$$

$$(\mu_0, \mu_1, \cdots, \mu_{254}) = \boldsymbol{\mu}, \text{ erasure vector}$$

$$(e_0, e_1, \cdots, e_{254}) = \mathbf{e}, \text{ error vector}$$

$$(\tilde{\mu}_0, \tilde{\mu}_1, \cdots, \tilde{\mu}_{254}) = \tilde{\boldsymbol{\mu}}, \text{ new erasure vector}$$

These vectors are related by $\mathbf{r} = \mathbf{c} + \boldsymbol{\mu} + \mathbf{e}$ and $\tilde{\boldsymbol{\mu}} = \mathbf{e} + \boldsymbol{\mu}$.

Suppose that t errors and s erasures occur in the received vector $\mathbf{r}$ of 255 symbols and assume $2t + s < 33$. Then the decoding procedure consists of the following steps:

Step 1: Compute the syndromes S_k $(1 \leqslant k \leqslant 32)$ of the received 255-tuple $(r_0, r_1, \cdots, r_{254})$, i.e.,

$$S_k = \sum_{i=0}^{254} r_i \alpha^{ik} \quad \text{for } k = 1, 2, \cdots, 32 \tag{1}$$

where α is an element of order 255 in $GF(2^8)$. If $S_k = 0$ for $1 \leqslant k \leqslant 32$, then $\mathbf{r}$ is a code word and no decoding is necessary. Otherwise,

Step 2: Compute τ_j for $j = 0, 1, 2, \cdots, s$ from the erasure locator polynomial

$$\tau(x) = \prod_{j=1}^{s} (x - Z_j) = \prod_{j=1}^{s} (-1)^j \tau_j x^{s-j} \tag{2}$$

where s is the number of erasures in the received vector, and Z_j $(1 \leqslant j \leqslant s)$ are the known erasure locations. Then compute the Forney syndromes T_i for $1 \leqslant i \leqslant d - 1 - s$ from the equation

$$T_i = \sum_{j=0}^{s} (-1)^j \tau_j S_{i+s-j} \quad \text{for } 1 \leqslant i \leqslant d - 1 - s \tag{3}$$

where τ_j $(1 \leqslant j \leqslant s)$ and S_j $(1 \leqslant j \leqslant 32)$ are known.

Step 3: If $0 \leqslant s < 32$, then use continued fractions to determine the error locator polynomial $\sigma(x)$ from the known T_i's $(1 \leqslant i < 32 - s)$. For the special case $s = 32$, it was shown (Ref. 3) that it is impossible for any decoder to tell whether there are zero or more additional errors. Thus, for $s = 32$, the best policy is not to decode the message at all. If $2t + s \geqslant 33$, it is shown in Ref. 4 that the continued fraction algorithm will not determine the correct error locator polynomial and will (1) either terminate abnormally, (2) terminate normally with a polynomial whose roots do not represent possible error locations, or (3) terminate normally with a polynomial whose roots do represent possible error locations.

In the first two cases, a decoding failure will occur; the decoder will be unable to decode the received word. In the third case, a decoding error will occur; there will be no alarm telling of the inability to decode; the decoder will operate as is $2t + s < 33$. The probability of such miscorrections of errors and erasures of R-S code words is discussed further in Ref. 4.

Step 4: Use a Chien-type search to find the t roots of the error locator polynomial. If t distinct roots cannot be found that represent possible error locations, then declare a decoding failure. Otherwise,

Step 5: Compute the combined erasure and error locator polynomial from the equation

$$\tilde{\tau}(x) = \sigma(x)\tau(x) = \sum_{k=0}^{s+t} (-1)^k \tilde{\tau}_k x^{s+t-k} \tag{4}$$

where $\sigma(x)$ and $\tau(x)$ are now known. Then compute the rest of transform of the erasure and error vector from the equation

$$S_\ell = \sum_{k=1}^{s+t} (-1)^k \tilde{\tau}_k S_{\ell-k} \quad \text{for } \ell > d - 1$$

Step 6: Invert the transform of $\tilde{\mu}$ at the points corresponding to the known error and erasure locations to obtain the amplitudes of $\tilde{\mu}$. That is,

$$\tilde{\mu}(Z_j) = e_i + \mu_i = \sum_{k=0}^{254} S_k Z_j^k \quad \text{for } j = 1, 2, \cdots, s + t \tag{5}$$

where Z_j are the known error and erasure locations of $\tilde{\mu}$. Then subtract from the received word the error and erasure vector to obtain the corrected code word.

III. Program Design and Implementation

The decoding procedure described in the previous section was implemented on the UNIVAC 1108 computer using FORTRAN V. This program was used to correct any combination of t errors and s erasures occurring in the 255-symbol R-S code words, where $2t + s < 33$. The overall basic structure of the program is given in Fig. 1. It is divided into a main program and five major subroutines.

The Main Program: This is the main driver of the rest of the program. It initializes the decoding process and keeps track of the elapsed CPU time.

Input: This subroutine generates a random code vector (polynomial) $R(x)$ for the R-S decoder and then adds errors and erasures $E(x)$ to it.

Step 1: The first 32 syndromes of the received vectors as well as the corrected vectors are calculated in this subroutine. In case the corrected received word is not an R-S code word, the subroutine will output the message, "The corrected received vector is not a codeword." This is helpful in confirming the correctness of the program, as well as indicating that the number of errors and erasures have exceeded the limits allowable by the decoder.

Step 2: This subroutine computes the Forney syndrome vector T (Eq. 3) from the erasure vector Z. The erasure locator polynomial $\tau(x)$ (Eq. 2) is also calculated here.

Step 3: The error locator polynomial $\sigma(x)$ is calculated from the Forney syndrome vector T using the continued fraction algorithm. The product of the error locator polynomial $\sigma(x)$ and the erasure locator polynomial $\tau(x)$ is next computed (Eq. 4). The coefficients of this erasure and error locator polynomial are used to compute the remaining terms $S_{33}, \cdots, S_{255}$. Also, the locations of the errors are now determined. (The erasure locations are known a priori by definition.)

Step 4: This step directly computes the inverse Fourier transform of the vector $(S_1, S_2, \cdots, S_{255})$ to obtain the nonzero error and erasure magnitudes. Finally, the received vector is corrected to provide an estimate of transmitted code word.

IV. Simulation Results

The computation times for decoding numerous code words that were corrupted by errors and erasures are given in Table 1. These results were obtained by computing each entry in Table 1 five times and then averaging. Along any row or column, the computation times tend to increase with the row or column indices until decoding failures occur due to an excess of allowable errors and erasures.

V. Summary

A software Reed-Solomon decoder has been developed for the (255, 223, 33) R-S code whose symbols belong to $GF(2^8)$. The simulation indicates that the decoder correctly reconstructs the transmitted code word from the received word if the code word has been corrupted by t errors and s erasures, where $2t + s < 33$.

Acknowledgment

This work was supported in part by the U.S. Air Force Office of Scientific Research under Grant Number AFOSR 75-2798 and also in part by NASA Contract NAS7-100.

References

1. I. S. Reed and T. K. Truong, "A Simplified Algorithm for Correcting Both Errors and Erasures of R-S Codes," in *The Deep Space Network Progress Report 42-48,* Jet Propulsion Laboratory, Pasadena, California, September 1978.
2. J. Odenwalder, et al., *Hybrid Coding Systems Study Final Report* Linkabit Corp., NASA CR114, 486, September 1972.
3. E. R. Berlekamp and J. L. Ramsey "Readable Erasures Improve the Performance of Reed-Solomon Codes" *IEEE Transactions on Information Theory*, Vol. IT-24, No. 5, Sept. 1978.
4. R. L. Miller, I. S. Reed, and T. K. Truong, "The Probability of Incorrectly Decoding Errors and Erasures and Reed-Solomon Code Words," submitted to *IEEE Trans. on Inform Theory.*

Table 1. Decoder execution times in seconds

	Errors																	
Erasures	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0.314	0.354	0.397	0.463	0.529	0.483	0.515	0.564	0.608	0.681	0.750	0.760	0.734	0.812	0.777	0.940	0.878	0.592
1	0.342	0.375	0.466	0.472	0.483	0.495	0.549	0.584	0.625	0.661	0.709	0.771	0.746	0.792	0.887	0.867	0.414	
2	0.377	0.427	0.439	0.455	0.485	0.557	0.561	0.616	0.609	0.737	0.813	0.703	0.811	0.899	0.871	0.837	0.577	
3	0.382	0.421	0.455	0.481	0.511	0.571	0.574	0.607	0.744	0.757	0.697	0.779	0.814	0.840	0.836	0.429		
4	0.439	0.456	0.466	0.524	0.594	0.634	0.588	0.629	0.761	0.679	0.710	0.778	0.848	0.943	0.956	0.577		
5	0.437	0.471	0.483	0.582	0.583	0.596	0.609	0.638	0.763	0.759	0.782	0.810	0.868	1.054	0.409			
6	0.439	0.472	0.552	0.572	0.574	0.631	0.696	0.656	0.697	0.724	0.808	0.794	0.930	0.997	0.521			
7	0.474	0.492	0.625	0.612	0.588	0.660	0.782	0.679	0.770	0.817	0.790	0.815	0.943	0.401				
8	0.492	0.610	0.647	0.661	0.640	0.648	0.699	0.692	0.820	0.897	0.784	0.864	0.906	0.521				
9	0.583	0.585	0.665	0.626	0.625	0.664	0.720	0.741	0.781	0.840	0.812	0.918	0.432					
10	0.659	0.607	0.668	0.683	0.730	0.724	0.746	0.767	0.769	0.823	0.936	1.034	0.485					
11	0.561	0.695	0.682	0.661	0.741	0.763	0.733	0.770	0.818	0.811	0.849	0.405						
12	0.606	0.661	0.681	0.756	0.691	0.741	0.753	0.783	0.919	0.830	0.912	0.462						
13	0.656	0.638	0.687	0.737	0.708	0.740	0.800	0.816	0.901	0.858	0.356							
14	0.620	0.760	0.716	0.740	0.735	0.779	0.785	0.831	0.924	0.949	0.449							
15	0.693	0.778	0.719	0.749	0.840	0.863	0.824	0.981	0.883	0.424								
16	0.662	0.735	0.715	0.766	0.758	0.860	0.833	0.907	0.912	0.481								
17	0.802	0.744	0.712	0.800	0.789	0.852	0.841	0.864	0.402									
18	0.858	0.720	0.756	0.825	0.806	0.980	0.859	0.879	0.506									
19	0.852	0.747	0.771	0.791	0.802	0.963	0.897	0.347										
20	0.756	0.740	0.841	0.935	0.831	0.878	0.972	0.428										
21	0.758	0.763	0.954	0.911	0.918	0.899	0.386											
22	0.750	0.805	0.956	0.898	0.875	1.049	0.406											
23	0.769	0.807	0.825	0.858	0.910	0.429												
24	0.811	0.902	0.893	0.892	0.907	0.439												
25	0.829	0.857	0.906	0.945	0.342													
26	0.872	0.920	0.907	0.928	0.372													
27	0.960	0.960	0.935	0.340														
28	0.914	0.908	0.935	0.507														
29	1.028	0.930	0.374															
30	0.985	0.945	0.826															
31	0.960	0.374																

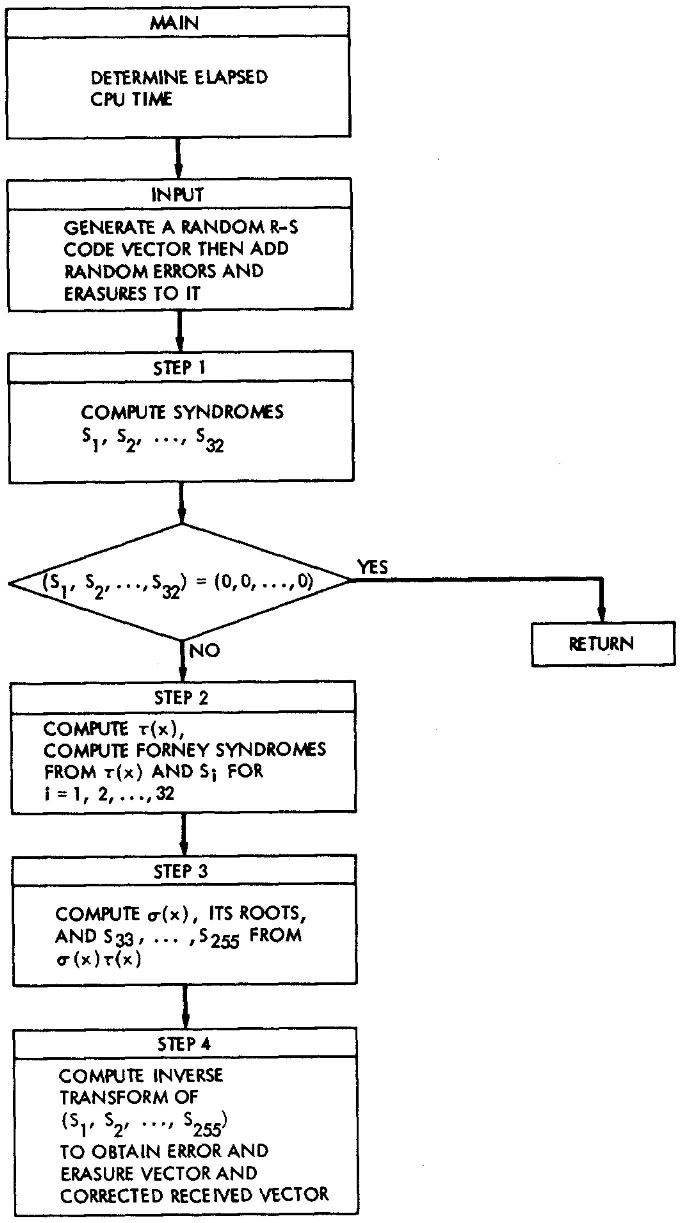
MAIN
DETERMINE ELAPSED CPU TIME
INPUT
GENERATE A RANDOM R-S CODE VECTOR THEN ADD RANDOM ERRORS AND ERASURES TO IT
STEP 1
COMPUTE SYNDROMES $S_1, S_2, \ldots, S_{32}$
$(S_1, S_2, \ldots, S_{32}) = (0, 0, \ldots, 0)$
YES
RETURN
NO
STEP 2
COMPUTE $\tau(x)$, COMPUTE FORNEY SYNDROMES FROM $\tau(x)$ AND S_i FOR $i = 1, 2, \ldots, 32$
STEP 3
COMPUTE $\sigma(x)$, ITS ROOTS, AND $S_{33}, \ldots, S_{255}$ FROM $\sigma(x)\tau(x)$
STEP 4
COMPUTE INVERSE TRANSFORM OF $(S_1, S_2, \ldots, S_{255})$ TO OBTAIN ERROR AND ERASURE VECTOR AND CORRECTED RECEIVED VECTOR

Fig. 1. Basic functional structures of R-S decoding program using transform over GF (2^8) and continued fractions